Chapter

How to Solve the Traveling Salesman Problem

Weiqi Li

Abstract

The Traveling Salesman Problem (TSP) is believed to be an intractable problem and have no practically efficient algorithm to solve it. The intrinsic difficulty of the TSP is associated with the combinatorial explosion of potential solutions in the solution space. When a TSP instance is large, the number of possible solutions in the solution space is so large as to forbid an exhaustive search for the optimal solutions. The seemingly "limitless" increase of computational power will not resolve its genuine intractability. Do we need to explore all the possibilities in the solution space to find the optimal solutions? This chapter offers a novel perspective trying to overcome the combinatorial complexity of the TSP. When we design an algorithm to solve an optimization problem, we usually ask the critical question: "How can we find all exact optimal solutions and how do we know that they are optimal in the solution space?" This chapter introduces the Attractor-Based Search System (ABSS) that is specifically designed for the TSP. This chapter explains how the ABSS answer this critical question. The computing complexity of the ABSS is also discussed.

Keywords: combinatorial optimization, global optimization, heuristic local search, computational complexity, traveling salesman problem, multimodal optimization, dynamical systems, attractor

1. Introduction

The TSP is one of the most intensively investigated optimization problems and often treated as the prototypical combinatorial optimization problem that has provided much motivation for design of new search algorithms, development of complexity theory, and analysis of solution space and search space [1, 2]. The TSP is defined as a complete graph $Q = (V, E, C)$, where $V = \{v_i : i = 1, 2, \ldots, n\}$ is a set of n nodes, $E = \{e(i,j) : i, j = 1, 2, \ldots, n; i \neq j\}_{n \times n}$ is an edge matrix containing the set of edges that connects the n nodes, and $C = \{c(i,j) : i, j = 1, 2, \ldots, n; i \neq j\}_{n \times n}$ is a cost matrix holding a set of traveling costs associated with the set of edges. The solution space S contains a finite set of all feasible tours that a salesman may traverse. A tour $s \in S$ is a closed route that visits every node exactly once and returns to the starting node at the end. Like many real-world optimization problems, the TSP is inherently multimodal; that is, it may contain multiple optimal tours in its solution space. We assume that a TSP instance Q contains h (≥ 1) optimal tours in S. We denote $f(s)$ as the objective function, $s^* = \min_{s \in S} f(s)$ as an optimal tour and S^* as the set of h optimal tours. The objective of the TSP is to find all h optimal tours in the solution space, that is, $S^* \subset S$. Therefore, the argument is

$$\arg\left[\min_{s\in S} f(s)\right] = S^* = \left[s_1^*, s_2^*, \ldots, s_h^*\right] \quad (1)$$

Under this definition, the salesman wants to know what all best alternative tours are available. Finding all optimal solutions is the essential requirement for an optimization search algorithm. In practice, knowledge of multiple optimal solutions is extremely helpful, providing the decision-maker with multiple options, especially when the sensitivity of the objective function to small changes in its variables may be different at the alternative optimal points. Obviously, this TSP definition is elegantly simple but full of challenge to the optimization researchers and practitioners.

Optimization has been a fundamental tool in all scientific and engineering areas. The goal of optimization is to find the best set of the admissible conditions to achieve our objective in our decision-making process. Therefore, the fundamental requirement for an optimization search algorithm is to find *all optimal solutions* within a *reasonable amount of computing time*. The focus of computational complexity theory is to analyze the intrinsic difficulty of an optimization problem and the asymptotic property of a search algorithm to solve it. The complexity theory attempts to address this question: "How efficient is a search algorithm for a particular optimization problem, as the number of variables gets large?"

The TSP is known to be NP-hard [2, 3]. The problems in NP-hard class are said to be intractable because these problems have no asymptotically efficient algorithm, even the seemingly "limitless" increase of computational power will not resolve their genuine intractability. The intrinsic difficulty of the TSP is that the solution space increases exponentially as the problem size increases, which makes the exhaustive search infeasible. When a TSP instance is large, the number of possible tours in the solution space is so large to forbid an exhaustive search for the optimal tours. A feasible search algorithm for the TSP is one that comes with a guarantee to find all best tours in time at most proportional to n^k for some power k.

Do we need to explore all the possibilities in the solution space to find the optimal solutions? Imagine that searching for the optimal solution in the solution space is like treasure hunting. We are trying to hunt for a hidden treasure in the whole world. If we are "blindfolded" without any guidance, it is a silly idea to search every single square inch of the extremely large space. We may have to perform a random search process, which is usually not effective. However, if we are able to use various clues to locate the small village where the treasure was placed, we will then directly go to that village and search every corner of the village to find the hidden treasure. The philosophy behind this treasure-hunting case for optimization is that: if we do not know where the optimal point is in the solution space, we can try to identify the small region that contains the optimal point and then search that small region thoroughly to find that optimal point.

Optimization researchers have developed many optimization algorithms to solve the TSP. Deterministic approaches such as exhaustive enumeration and branch-and-bound can find exact optimal solutions, but they are very expensive from the computational point of view. Stochastic optimization algorithms, such as simple heuristic local search, Evolutionary Algorithms, Particle Swarm Optimization and many other metaheuristics, can find hopefully a good solution to the TSP [1, 4–7]. The stochastic search algorithms trade in guaranteed correctness of the optimal solution for a shorter computing time. In practice, most stochastic search algorithms are based on the heuristic local search technique [8]. Heuristics are functions that help us decide which one of a set of possible solutions is to be selected next [9]. A local search algorithm iteratively explores the neighborhoods of solutions trying to improve the current solution by a local change. However, the scope of local search is

limited by the neighborhood definition. Therefore, heuristic local search algorithms are locally convergent. The final solution may deviate from the optimal solution. Such a final solution is called a *locally optimal solution*, denoted as s' in this chapter. To distinguish from locally optimal solutions, the optimal solution s^* in the solution space is usually called the *globally optimal solution*.

This chapter studies the TSP from a novel perspective and presents a new search algorithm for the TSP. This chapter is organized in the following sections. Section 2 presents the ABSS algorithm for the TSP. Section 3 describes the important data structure that is a critical player in solving the TSP. Section 4 discusses the nature of heuristic local search algorithm and introduces the concept of solution attractor. Section 5 describes the global optimization features of the ABSS. Section 6 discusses the computational complexity of the ABSS. Section 7 concludes this chapter.

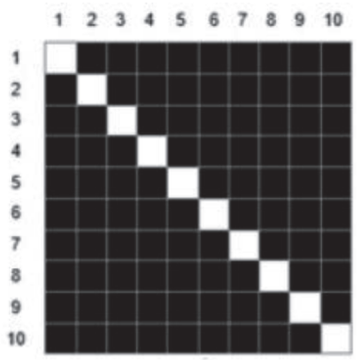
2. The attractor-based search system for the TSP

Figure 1 presents the Attractor-Based Search System (ABSS) for the TSP. In this algorithm, Q is a TSP instance with the edge matrix E and cost matrix C. At beginning of search, the matrix E is initialized by assigning zeros to all elements of E. The function `InitialTour()` constructs an initial tour s_i using any tour-construction technique. The function `LocalSearch()` takes s_i as an input, performs local search using any type of local search technique, and returns a locally optimal tour s_j. The function `UpdateE()` updates the matrix E by recording the edge configuration of tour s_j into the matrix. K is the number of search trajectories. After the edge configurations of K locally optimal tours are stored in the matrix E, the function `ExhaustedSearch()` searches E completely using the depth-first tree search technique, which is a simple recursive search method that traverses a directed graph starting from a node and then searches adjacent nodes recursively. Finally, the ABSS outputs a set of all best tours S^* found in the edge configuration of E. The search strategy in the ABSS is straightforward: generating K locally optimal tours, storing their edge configurations in the matrix E, and then identifying the best tours by evaluating all tours represented by the edge configuration of E. The ABSS is a simple and efficient computer program that can solve the TSP effectively. This search algorithm shows strong features of effectiveness, flexibility, adaptability, scalability and efficiency. The computational model of the ABSS is inherently parallel, facilitating implementation on concurrent processors. It can be implemented in many different ways: series, parallel, distributed, or hybrid.

```
1   ABSS(Q)
2   begin
3     Initialize E;
4     NumberOfTrajectories = 0;
5     repeat
6       s_i = InitialTour();
7       s_j = LocalSearch(s_i);
8       E = UpdateE(s_j, E);
9       NumberOfTrajectories = NumberOfTrajectories + 1;
10    until NumberOfTrajectories = K
11    S* = ExhaustedSearch(E);
12  end
```

Figure 1.
The ABSS algorithm for the TSP.

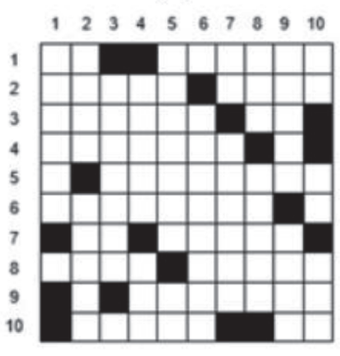
Figure 2 uses a 10-node instance as an example to illustrate how the ABSS works. We randomly generate $K = 6n = 60$ initial tours, which edge configurations hit all elements of the matrix E (marked as black color), as shown in **Figure 2(a)**. It means that these 60 random tours hit all 45 edges that represent all 181440 tours in the solution space. We let each of the search trajectories run 5000 iterations and obtain 60 locally optimal tours. However, due to the small size of the instance, most locally optimal tours have identical edge configurations. Among the 60 locally optimal tours, we find only four distinct locally optimal tours as shown in **Figure 2(b)**. **Figure 2(c)** shows the union of the edge configurations of the 60 locally optimal tours, in which 18 edges are hit. Then we use the depth-first tree search, as illustrated in **Figure 2(d)**, to identify all five tours in the edge configuration of E, which are listed in **Figure 2(e)**. In fact, one of the five tours is the globally optimal tour. This simple example indicates that (1) local search trajectories converge to small set of edges, and (2) the union of the edge configurations of K locally optimal tours is not just a countable union of the edge configurations of the these tours, but also include the edge configurations of other locally optimal tours. The ABSS consists of two search phases: local search phase and exhaustive search phase.



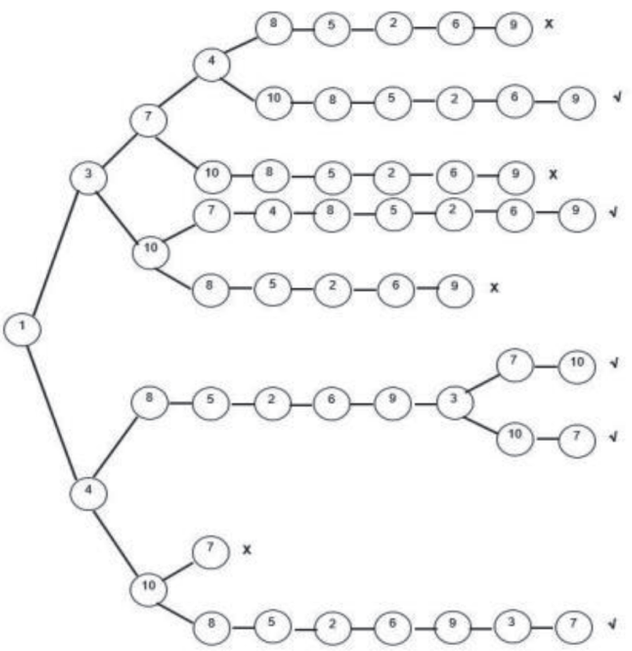
(a)

(1, 3, 7, 4, 10, 8, 5, 2, 6, 9)
(1, 4, 8, 5, 2, 6, 9, 3, 7, 10)
(1, 4, 8, 5, 2, 6, 9, 3, 10, 7)
(1, 4, 10, 8, 5, 2, 6, 9, 3, 7)

(b)



(c)



(d)

(1, 3, 7, 4, 10, 8, 5, 2, 6, 9)
(1, 4, 8, 5, 2, 6, 9, 3, 7, 10)
(1, 4, 8, 5, 2, 6, 9, 3, 10, 7)
(1, 4, 10, 8, 5, 2, 6, 9, 3, 7)

(1, 3, 10, 7, 4, 8, 5, 2, 6, 9)

(e)

Figure 2.
A simple example of the ABSS algorithm. (a) Union of the edge configurations of 60 random initial tours, (b) four distinct locally optimal tours, (c) union of the edge configurations of the 60 locally optimal tours, (d) the depth-first tree search on the edge configuration of E*, and (e) five tours found in* E.

The task of the local search phase is to identify the region that globally optimal tour is located (i.e. the village hiding the treasure), and the task of the exhaustive search phase is to find the globally optimal tour (i.e. find the hidden treasure). The remaining sections will briefly explain the features of the ABSS.

In all experiments mentioned in the chapter, we generate symmetric TSP instances with n nodes. The element $c(i,j)$ of the cost matrix C is assigned a random integer independently drawn from a uniform distribution of the range [1, 1000]. The triangle inequality $c(i,j)+c(j,k)\geq c(i,k)$ is not assumed in the instances. Although this type of problem instances is application-free, it is mathematically significant. A TSP instance without triangle inequality cannot be approximated within any constant factor. A heuristic local search algorithm usually performs much worse for this type of TSP instances, which offers a strikingly challenge to solving them [2, 3, 6, 10, 11]. We use the 2-opt local search technique in the local search phase. The 2-opt neighborhood can be characterized as the neighborhood that induces the greatest correlation between function values of neighboring tours, because neighboring tours differ in the minimum possible four edges. Along the same reasoning line, the 2-opt may have the smallest expected number of locally optimal points [12]. The local search process randomly selects a solution in the neighborhood of the current solution. A move that gives the first improvement is chosen. The great advantage of the first-improvement pivoting rule is to produce randomized locally optimal points. The software program written for the experiments use several different programming languages and are run in PCs with different versions of Window operating system.

3. The edge matrix *E*

Usually the edge matrix E is not necessary to be included in the TSP definition because the TSP is a complete graph. However, the edge matrix E is an effective data structure that can help us understand the search behavior of a local search system. General local search algorithm may not require much problem-specific knowledge in order to generate good solutions. However, it may be unreasonable to expect a search algorithm to be able to solve any problem without taking into account the data structure and properties of the problem at hand.

To solve a problem, the first step is to create a manipulatable description of the problem itself. For many problems, the choice of data structure for representing a solution plays a critical role in the analysis of search behavior and design of new search algorithm. For the TSP, a tour can be represented by an ordered list of nodes or an edge configuration of a tour in the edge matrix E, as illustrated in **Figure 3**. The improvement of the current tour represents the change in the order of the nodes or the edge configuration of a tour.

Observing the behavior of search trajectories in a local search system can be quite challenging. The edge matrix E is a natural data structure that can help us trace the search trajectories and understand the dynamics of a local search system. An edge $e(i,j)$ is the most basic element of a tour, but contains a piece of information about each of $(n-2)!$ tours that go through it. Essentially, the nature of local search for the TSP is an edge-selection process: preservation of good edges and rejection of bad edges according to the objective function $f(s)$. Each edge has an implicit probability to be selected by a locally optimal tour. A better edge has higher probability to be included in a locally optimal tour. Therefore, the edges in E can be divided into three groups: globally superior edges, G-edges, and bad edges. A globally superior edge is the edge that occurs in many or all locally optimal tours. Although each of these locally optimal tours selects this edge based on its own

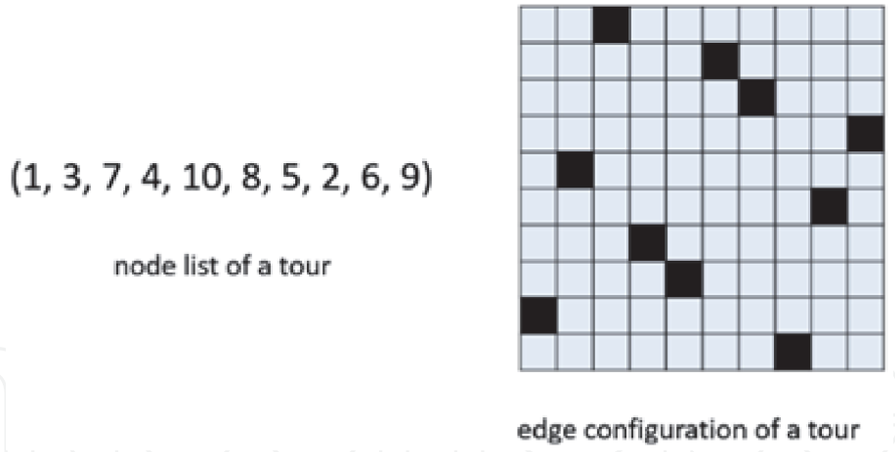


Figure 3.
Two representations of a tour: an ordered list of nodes and an edge configuration of a tour.

search trajectory, the edge is globally superior since the edge is selected by these individual tours from different search trajectories going through different search regions. The globally superior edges have higher probability to be selected by a locally optimal tour. A *G*-edge is the edge that is included in a globally optimal tour. All *G*-edges are globally superior edges and can be treated as a special subset of the globally superior edges. The edges that are discarded by all search trajectories or selected by only few locally optimal tours are bad edges. A bad edge is impossible to be included in a globally optimal tour. A locally optimal tour usually consists of some *G*-edges, some globally superior edges and a few bad edges.

The changes of the edge configuration of the matrix E represent the transformations of the search trajectories in a local search system. When all search trajectories reach their end points, the final edge configuration of E represents the final state of the local search system. For a tour s_k, we define an element $e(i,j)$ of E as

$$e(i,j) = \begin{cases} 1 \text{ if the element } e(i,j) \text{ is in the tour } s_k \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

Then the hit-frequency value e_{ij} in the element $e(i,j)$ is defined as the number of occurrence of the element in K tours, that is

$$e_{ij} = \sum_{k=1}^{K} e(i,j)_k \tag{3}$$

When K search trajectories reach their end points, the value $\left(e_{ij} + e_{ji}\right)/K$ can represent the probability of the edge $e(i,j)$ being hit by a locally optimal tour. We can use graphical technique to observe the convergent behavior of the search trajectories through the matrix E. The hit-frequency value e_{ij} can be easily converted into a unit of half-tone information in a computer, a value that we interpret as a number H_{ij} somewhere between 0 and 1. The value 1 corresponds to black color, 0 to white color, and any value in between to a gray level. Let K be the number of search trajectories, the half-tone information H_{ij} on a computer screen can be represented by the hit-frequency e_{ij} in the element $e(i,j)$ of E:

$$H_{ij} = \frac{e_{ij}}{K} \tag{4}$$

Figure 4 illustrates a simple example of visualization showing the convergent behavior of 100 search trajectories for a 50-node instance. **Figure 4(a)** shows the image of the edge configurations of 100 random initial tours. Since each element of E has equal chance to be hit by these initial tours, almost all elements are hit by these initial tours, and all elements have very low H_{ij} values, ranging from 0.00 to 0.02. When the local search system starts searching, the search trajectories constantly change their edge configurations, and therefore the colors in the elements of E are changed accordingly. As the search continues, more and more elements become white (i.e. they are discarded by all search trajectories) and other elements become darker (i.e. they are selected by more search trajectories). When all search trajectories reach their end points, the colored elements represent the final edge configuration of the search system. **Figure 4(b)** and **(c)** show the images of edge configuration of E when all search trajectories completed 2000 iterations and 5000 iterations, respectively. At 5000th iteration, the range of H_{ij} values in the elements of E is from 0.00 to 0.42. The value 0.42 means that 42% of the search trajectories select this element. Majority of the elements of E become white color.

This simple example has great explanatory power about the global dynamics of the local search system for the TSP. As search trajectories continue searching, the number of edges hit by them becomes smaller and smaller, and better edges are hit by more and more search trajectories. This edge-convergence phenomenon means that all search trajectories are moving closer and closer to each other, and their edge configurations become increasingly similar. This phenomenon describes the globally asymptotic behavior of the local search system.

It is easily verified that under certain conditons, a local search system is able to find the set of the globally optimal tours S^* when the number of search trajectories is unlimited, i.e.

$$\lim_{K\to\infty} P[S^* \subset S] = 1 \tag{5}$$

However, the required search effort may be very huge – equivalent to enumerating all tours in the solution space. Now one question for the ABSS is "How many search trajectories in the search system do we need to find all globally optimal tours?" The matrix E consists of $n(n-1)$ elements (excluding the diagonal elements). When we randomly construct a tour and record its edge configuration in E, n elements of E will be hit by this tour. If we construct more random tours and record their edge configurations in E, more elements will be hit. We define K as the number of randomly-constructed initial tours, whose edge configurations together

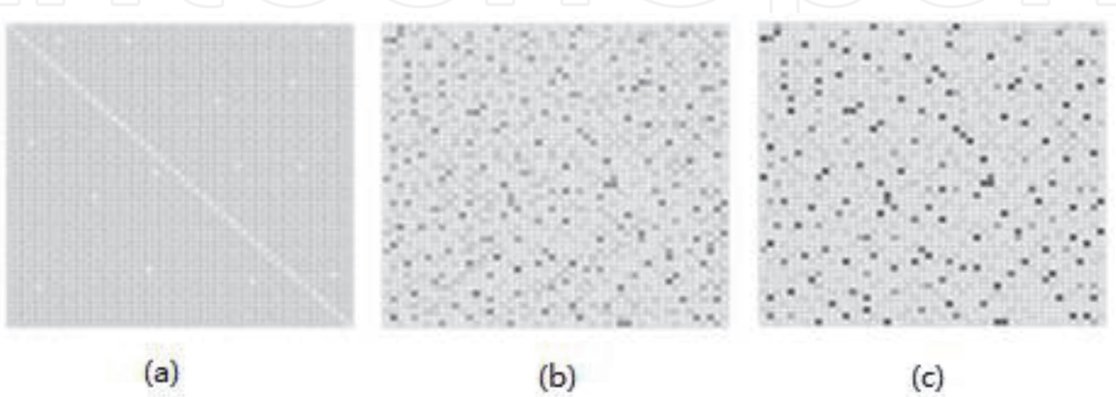


Figure 4.
Visualization of the convergent dynamics of local search system. (a) the image of the edge configurations of 100 initial tours, (b) and (c) the images of edge configurations when the search trajectories are at 2000th and 5000th iteration, respectively.

will hit all elements of E. We know that all elements of E represent all combinatorial possibilities in the solution space. Therefore, K is the number of search trajectories such that the union of edge configurations of ther initial tours covers the entire solution space. In our experiments, we found that the edge configurations of at most $6n$ randomly-constructed tours can guarantee to hit all elements of E. From the tour perspective, $K = 6n$ random tours represent only a small set of the tours in the solution space. However, from the view of edge-configuration, the union of the edge configurations of $6n$ random tours represents the edge configurations of all tours in the solution space. It reveals an amazing fact: the union of the edge configurations of only $6n$ random tours contains the edge configurations of all $n(n-1)!/2$ tours in the solution space. It reflects the combinatorial nature of the TSP: the tours in the solution space are formed by different combinations of the edges. The union of the edge configurations of a set of tours contains information about many other tours because one tour shares its edges with many other tours. One fundamental theory that can help us explain this phenomenon is the information theory [13]. According to the information theory, each solution point contains some information about its neighboring solutions that can be modeled as a function, called *information function* or *influence function*. The influence function of the i^{th} solution point in the solution space S is defined as a function $\Omega_i : S \rightarrow \Re$, such that Ω_i is a decreasing function of the distance from a solution point to the i^{th} solution point. The notion of influence function has been extensively used in datamining, data clustering, and pattern recognition.

4. The nature of heuristic local search

Heuristic local search is based on the concept of neighborhood search. A neighborhood of a solution s_i, denoted as $N(s_i)$, is a set of solutions that are in some sense close to s_i. For the TSP, a neighborhood of a tour s_i is defined as a set of tours that can be reached from s_i in one single transition. From edge-configuration perspective, all tours in $N(s_i)$ are very similar because they share significant number of edges with s_i. The basic operation of local search is iterative improvement, which starts with an initial tour and searches the neighborhood of the current tour for a better tour. If such a tour is found, it replaces the current tour and the search continues until no improvement can be made. The local search algorithm returns a locally optimal tour.

The behavior of a local search trajectory can be understood as a process of iterating a search function $g(s)$. We denote s_0 as an initial point of search and $g^t(s)$ as the t^{th} iteration of the search function $g(s)$. A search trajectory $s_0, g(s_0), g^2(s_0), \ldots, g^t(s_0), \ldots$ converges to a locally optimal point s' as its limit, that is,

$$g\left(\lim_{t\rightarrow\infty} g^t(s_0)\right) = \lim_{t\rightarrow\infty} g^{t+1}(s_0) = s' \tag{6}$$

Therefore, a search trajectory will reach an end point (a locally optimal point) and will stays at this point forever.

In a heuristic local search algorithm, there is a great variety of ways to construct initial tour, choose candidate moves, and define criteria for accepting candidate moves. Most heuristic local search algorithms are based on randomization. In this sense, a heuristic local search algoorithm is a randomized system. There are no two search trajectories that are exactly alike in such a search system. Different search

trajectories explore different regions of the solution space and stop at different final points. Therefore, local optimality depends on the initial points, the neighborhood function, randomness in the search process, and time spent on search process. On the other hand, however, a local search algorithm essentially is deterministic and not random in nature. If we observe the motion of all search trajectories, we will see that the search trajectories go towards the same direction, move closer to each other, and eventually converge into a small region in the solution space.

Heuristic local search algorithms are essentially in the domain of dynamical systems. A heuristic local search algorithm is a discrete dynamical system, which has a solution space S (the state space), a set of times T (search iterations), and a search function $g : S \times T \rightarrow S$ that gives the consequents to a solution $s \in S$ in the form of $s_{t+1} = g(s_t)$. A search trajectory is the sequence of states of a single search process at successive time-steps, which represents the part of the solution space searched by this search trajectory. The questions about the behavior of a local search system over time are actually the questions about its search trajectories. The most basic question about the search trajectories is "Where do they go in the solution space and what do they do when they get there?"

The attractor theory of dynamical systems is a natural paradigm that can be used to describe the search behavior of a heuristic local search system. The theory of dynamical systems is an extremely broad area of study. A dynamical system is a model of describing the temporal evolution of a system in its state space. The goal of dynamical system analysis is to capture the distinctive properties of certain points or regions in the state space of a given dynamical system. The theory of dynamical systems has discovered that many dynamical systems exhibt attracting behavior in the state space [14–22]. In such a system, all initial states tend to evolve towards a single final point or a set of points. The term *attractor* is used to describe this single point or the set of points in the state space. The attractor theory of dynamical systems describes the asymptotic behavior of typical trajectories in the dynamical system. Therefore, the attractor theory provides the theoretical foundation to study the search behavior of a heuristic lcoal search system.

In a local search system for the TSP, no matter where we start a search trajectory in the solution space, all search trajectories will converge to a small region in the solution space for a unimodal TSP instance or h small regions for a h-model TSP. We call this small region a *solution attractor* of the local search system for a given TSP instance, denoted as A. Therefore, the solution attractor of a local search system for the TSP can be defined as an invariant set $A \subset S$ consisting of all locally optimal tours and the globally optimal tours. A single search trajectory typically converges to either one of the points in the solution attractor. A search trajectory that is in the solution attractor will remain within the solution attractor forward in time. Because a globally optimal tour s^* is a special case of locally optimal tours, it is undoubtedly embodied in the solutioin attractor, that is, $s^* \in A$. For a h-modal TSP instance, a local search system will generate h solution attractors $(A_1, A_2, \ldots, A_h)$ that attract all search trajectories. Each of the solution attractors has its own set of locally optimal tours, surrounding a globally optimal tour s_i^* $(i = 1, 2, \ldots, h)$. A particular search trajectory will converge into one of the h solution attractors. All locally optimal tours will be distributed to these solution attractors. According to dynamical systems theory [20], the closure of an arbitrary union of attractors is still an attractor. Therefore, the solution attractor A of a local search system for a h-modal TSP is a complete collection of h solution attractors $A = A_1 \cup A_2 \cup \ldots \cup A_h$.

The concept of solution attractor of local search system describes where the search trajectories actually go and where their final points actually stay in the solution space. **Figure 5** visually summarizes the concepts of search trajectories and solution attractors in a local search system for a multimodal optimization problem,

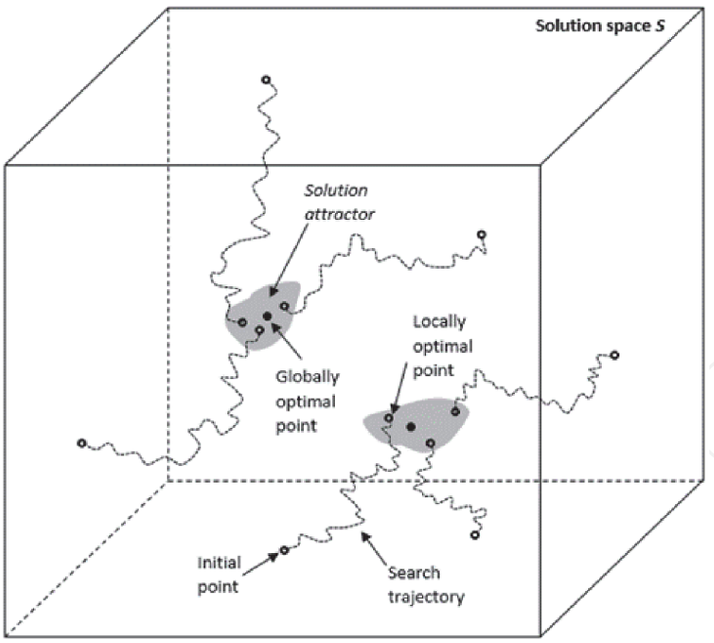


Figure 5.
Illustration of the concepts of serch trajectories and solution attractors in a local search system for a multimodal optimization problem.

describing how search trajectories converge and how solution attractors are formed. In summary, let $g(s)$ be a search function in a local search system for the TSP, the solution attractor of the search system has the following properties [23–25]:

1. *Convexity*, i.e. $\forall s_i \in S, g^t(s_i) \in A$ for sufficient long t;

2. *Centrality*, i.e. the globally optimal tour s_i^* is located centrally with respect to the other locally optimal tours in A_i $(i = 1, 2, \ldots h)$;

3. *Invariance*, i.e. $\forall s' \in A, g^t(s') = s'$ and $g^t(A) = A$ for all time t;

4. *Inreducibility*, i.e. the solution attractor A contains a limit number of invariant locally optimal tours.

A search trajectory in a local search system changes its edge configuration during the search according to the objective function $f(s)$ and its neighborhood structure. The matrix E can follow the "footprints" of search trajectories to capture the dynamics of the local search system. When all search trajectories reach their end points – the locally optimal tours, the edge configuration of the matrix E will become fixed, which is the edge configuration of the solution attractor A. This fixed edge configuration contains two groups of edges: the edges that are not hit by any of locally optimal tours (non-hit edges) and the edges that are hit by at least one of the locally optimal tours (hit edges). **Figure 6** shows the edge grouping in the edge configuration of E when all search trajectories stop at their final points.

In the ABSS, we use K search trajectories in the local search phase. Different sets of K search trajectories will generate different final edge configuration of E. Suppose that, we start the local search from a set of K initial points and obtain a edge configuration M_a in E when the local search phase is terminated. Then we start the local search process again from a different set of K initial points and obtains a little different edge configuration M_b in E. Which edge configuration truly describes the edge configuration of the real solution attractor? Actually, M_a and M_b are structurally

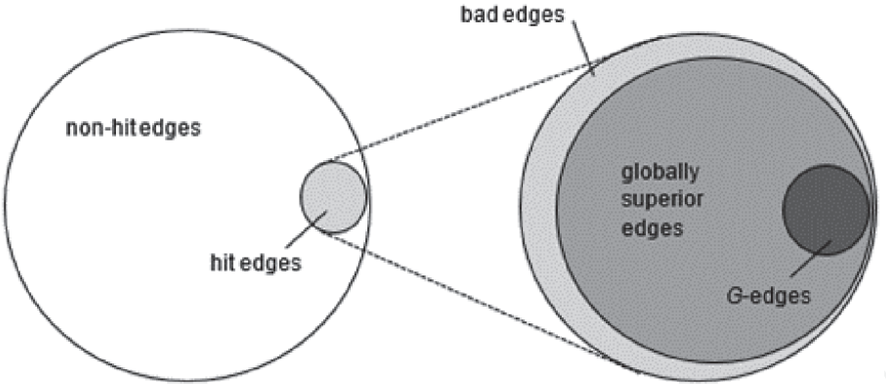


Figure 6.
The grouping of the edges in E *when all search trajectories reach their end points.*

equivalent because they are different only in the set of bad edges, thus M_a precisely replicates the dynamical properties of M_b. The final edge configuration of the constructed solution attractor generated from K search trajectories is not sensitive to the selection of K search trajectories. This property indicates that a heuristic local search system actually is a deterministic system: although a single search trajectory appears stochastic, all search trajectories from differeitn initial points will be always trapped into the same small region in the solution space and the final edge configuration of E will always converge to the same set of the globally optimal edges.

The convergence of the search trajectories can be measured by the change in the edge configuration of the matrix E. In the local search process, search trajectories collect all available topology information about the quality of the edges from their search experience and record such information in the matrix E. The changes in the edge configuration of E fully reflects the real search evolution of the search system. A state of convergence is achieved once no any more local search trajectory can change the edge configuration of E. For a set of search trajectories to be converging, they must be getting closer and closer to each other, that is, their edge configurations become increasingly similar. As a result, the edge configurations of the search trajectories converge to a small set of edges that contains all globally superior edges and some bad edges. Let W denote total number of edges in E, $\alpha(t)$ the number of the edges that are hit by all search trajectories at time t, $\beta(t)$ the number of the edges that are hit by one or some of the search trajectories, and $\gamma(t)$ the number of edges that have no hit at all, then at any time t, we have

$$W = \alpha(t) + \beta(t) + \gamma(t) \tag{7}$$

For a given TSP instance, W is a constant value $W = n(n-1)/2$ for a symmetric instance or $W = n(n-1)$ for an asymmetric instance. During the local search process, the values for $\alpha(t)$ and $\gamma(t)$ will increase and the value for $\beta(t)$ will decrease. However, these values cannot increase or decrease foreover. At certain point of time, they will become constant values, that is,

$$W = \lim_{t\to\infty}\alpha(t) + \lim_{t\to\infty}\beta(t) + \lim_{t\to\infty}\gamma(t) = \mathrm{A} + \mathrm{B} + \Gamma \tag{8}$$

Our experiments confirmed this inference about $\alpha(t)$, $\beta(t)$ and $\gamma(t)$. **Figure 7** illustrates the patterns of $\alpha(t)$, $\beta(t)$ and $\gamma(t)$ curves generated in our experiments. Our experiments also found that, for unimodal TSP instances, the ratio $\gamma(t)/W$ could approach to 0.70 quickly for different sizes of TSP instances. For multimodal TSP instances, this ratio depends on the number of the globally optimal points. However, the set of hit edges is still very small.

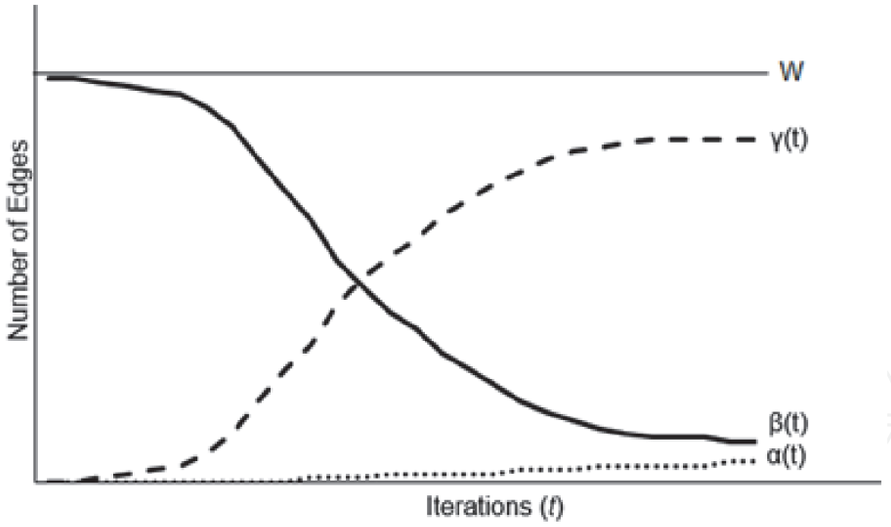


Figure 7.
The $\alpha(t)$, $\beta(t)$ and $\gamma(t)$ curves with search iterations.

In summary, we assume a TSP instance Q has a solution space with h (≥ 1) globally optimal tours $(s_1^*, s_2^*, \ldots, s_h^*)$, and correspondingly there exist h set of G-edges $(G_1, G_2, \ldots, G_h)$. A local search system for the Q will generate h solution attractors $(A_1, A_2, \ldots, A_h)$ that attract all search trajectories. The edge configuration of the solution attractor A is the union of the edge configurations of the h solution attractors. The final edge configuration of E represents the edge configuration of A with three properties:

1. It contains all locally optimal tours;

2. It contains a complete collection of solution attractors, i.e. $A = A_1 \cup A_2 \cup \ldots \cup A_h$;

3. It contains a complete collection of G-edges, i.e. $G = G_1 \cup G_2 \cup \ldots \cup G_h$.

From this analysis, we can see that the edge matrix E is an extremely useful data structure that not only collcets the information about search trajectories, but also convert local search behavor of individual search trajectories into global search behavor of the search system. The global convergence and deterministic property of the search trajectories make the local search system always converge to the same solution attractors and the edge configurations of the search trajectories always converge to the same set of globally superior edges. The matrix E shows us clearly where the search trajectories go and where all locally optimal points are located. We found the village! However, it is still difficult to identify all G-edges among the globally superior edges. The ABSS uses the exhaustive search phase to find all tours in the solution attractor. Since the local search phase has significantly reduced the size of the search space for the exhaustive search phase, the complete search in the solution attractor becomes feasible.

5. Global optimization feature of the ABSS

The task of a global optimization system is to find all absolutely best solutions in the solution space. There are two major tasks performed by a global optimization system: (1) finding all globally optimal points in the solution space and (2) making sure that they are globally optimal. So far we do not have any effective and efficient

global search algorithm to solve NP-hard combinatorial problems. We do not even have well-developed theory or analysis tool to help us design efficient algorithms to perform these two tasks. One critical question in global optimization is how to recognize the globally optimal solutions. Modern search algorithms lack practical criteria that decides when a locally optimal solution is a globally optimal one. What is the necessary and sufficient condition for a feasible point s_i to be globally optimal point? The mathematical condition for the TSP is $\forall s \in S, f(s^*) \leq f(s)$. To meet this condition, an efficient global search system should have the following properties:

1. The search system should be globally convergent.

2. The search system should be deterministic and have a rigorous guarantee for finding all globally optimal solutions without excessive computational burden.

3. The optimality criterion in the system must be based on information on the global behavior of the search system.

The ABSS combines beautifully two crucial aspects in search: exploration and exploitation. In the local search phase, K search trajectories explore the full solution space to identify the globally superior edges, which form the edge configuration of the solution attractor. These K search trajectories are independently and invidually executed, and therefore they create and maintain diversity from beginning to the end. The local search phase is a randomized process due to randomization in the local search function $g(s)$. In this sense, the K search trajectories actually perform the Monte Carlo simulation to sample locally optimal tours. The essential idea of Monte Carlo method is using randomness to solve problems that might be deterministic in principle [26]. In the ABSS, K search trajectories start a sample of initial points from a uniform distribution over the solution space S, and, through the randomized local search process, generate a sample of locally optimal points uniformly distributed in the solution attractor A. The edge configuration of E is actually constructed through this Monte Carlo sampling process.

Each of the K search trajectories passes through many neighborhoods on its way to the final point. For any tour s_i, the size of $N(s_i)$ is greater than $\left(\frac{n}{2}\right)!$ [12]. Let $N(s_i')$ denote the neighborhood of the final point s_i' of the i^{th} search trajectory and $\Omega N(s_{tran})_i$ as the union of the neighborhoods of all transition points of the search trajectory, then we can believe that the search space covered by K search trajectories is

$$N(s_1') \cup \Omega \mathrm{N}(s_{tran})_1 \cup \mathrm{N}(s_2') \cup \Omega \mathrm{N}(s_{tran})_2 \ldots \cup \mathrm{N}(s_K') \cup \Omega \mathrm{N}(s_{tran})_K = S \tag{9}$$

That is, the solution attractor A is formed through the entire solution space S. The solution attractor A contains h unique minimal "convex" sets A_i $(i = 1, 2, \ldots, h)$. Each A_i has a unique best tour s_i^* surrounded by a set of locally optimal tours. The tour s_i^* in A_i satisfies $f(s_i^*) < f(s)$ for all $s \in A_i$ and $f(s_1^*) = f(s_2^*) = \ldots = f(s_h^*)$.

We see that the matrix E plays a critical role to transform local search process of the individual search trajectories into a collective global search process of the system. Each time when a local search trajectory finds a better tour and updates the edge configuraton of E, the conditional distribution on the edges are updated. More values are attached to the globally superior edges, and bad edges are discarded. Let W be the complete set of the edges in E and W_A the set of edges in the edge configuration of the solution attractor A such that $g(W)$ is contained in the interior of W. Then the intersection W_A of the nested sequence of sets is

$$W \supset g(W) \supset g^2(W) \supset \ldots \supset g^t(W) \supset \ldots \supset W_A \tag{10}$$

and $\lim_{t\to\infty} g^t(W_A) = W_A$. As a result, the edge configurations of K search trajectories converge to a small set of edges.

The "convexity" property of the solution attractor A allows the propagation of the minimum property of s_i^* in the solution attractor A_i to the whole solution space S through the following conditions:

1. $\forall s \in A_i, f(s_i^*) < f(s)$

2. $f(s_1^*) = f(s_2^*) = \ldots = f(s_h^*)$

3. $\min_{s\in A} f(s) = \min_{s\in S} f(s)$

Therefore the global convergence and deterministic property of the search trajectories in the local search phase make the ABSS always find the same set of globally optimal tours. We conducted several experiments to confirm this argument empirically. In our experiments, for a given TSP instance, the ABSS performed the same search process on the instance several times, each time using a different set of K search trajectories. The ABSS outputed the same set of the best tours in all trials.

Table 1 shows the results of two experiments. One experiment generated $n = 1000$ instance Q_{1000}, the other generated $n = 10000$ instance Q_{10000}.

Trial number	Number of tours in A	Range of tour cost	Number of best tours in A
Q_{1000} (6000 search trajectories)			
1	6475824	[3241, 4236]	1
2	6509386	[3241, 3986]	1
3	6395678	[3241, 4027]	1
4	6477859	[3241, 4123]	1
5	6456239	[3241, 3980]	1
6	6457298	[3241, 3892]	1
7	6399867	[3241, 4025]	1
8	6423189	[3241, 3924]	1
9	6500086	[3241, 3948]	1
10	6423181	[3241, 3867]	1
Q_{10000} (60000 search trajectories)			
1	8645248	[69718, 87623]	4
2	8657129	[69718, 86453]	4
3	8603242	[69718, 86875]	4
4	8625449	[69718, 87053]	4
5	8621594	[69718, 87129]	4
6	8650429	[69718, 86978]	4
7	8624950	[69718, 86933]	4
8	8679949	[69718, 86984]	4
9	8679824	[69718, 87044]	4
10	8677249	[69718, 87127]	4

Table 1.
Tours in constructed solution attractor A *for* Q_{1000} *and* Q_{10000}.

We conducted 10 trials on each of the instances respectively. In each trial, the ABSS used $K = 6n$ search trajectories. Each search trajectory stopped when no improvement was made during $10n$ iterations. The matrix E stored the edge configurations of the K final tours and then was searched completely using the depth-first tree search process. **Table 1** lists the number of tours found in the constructed solution attractor A, the cost range of these tours, and the number of the best tours found in the constructed solution attractor. For instance, in trial 1 for Q_{1000}, the ABSS found 6475824 tours with the cost range [3241, 4136] in the constructed solution attractor. There was a single best tour in the solution attractor. The ABSS found the same best tour in all 10 trials. For the instance Q_{10000}, the ABSS found the same set of four best tours in all 10 trials. These four best tours have the same cost value, but with different edge configurations. If any trial had generated a different set of the best tours, we could immediately make a conclusion that the best tours in the constructed solution attractor may not be the globally optimal tours. From practical perspective, the fact that the same set of the best tours was detected in all trials provides an empirical evidence of the global optimality of these tours. The fact also indicates that the ABSS converges in solution. *Convergence in solution* means that the search system can identify all optimal solutions repeatedly. Always finding the same set of optimal solutions actually is the fundamental requirement for global optimization systems.

6. Computing complexity of the ABSS

With current search technology, the TSP is an infeasible problem because it is not solvable in a reasonable amount of time. Faster computers will not help. A feasible search algorithm for the TSP is one that comes with a guarantee to find all best tours in time at most proportional to n^k for some power k. The ABSS can guarantee to find all globally optimal tours for the TSP. Now the question is how efficient it is?

The core idea of the ABSS is that, if we have to use exhaustive search to confirm the globally optimal points, we should first find a way to quickly reduce the effective search space for the exhaustive search. When a local search trajectory finds a better tour, we can say that the local search trajectory finds some better edges. It is an inclusive view. We also can say that the local search trajectory discards some bad edges. It is an exclusive view. The ABSS uses the exclusive strategy to conquer the TSP. The local search phase in the ABSS quickly prunes out large number of edges that cannot possibly be included in any of the globally optimal tours. Thus, a large useless area of the solution space is excluded. When the first edge is discarded by all K search trajectories, $(n-2)!$ tours that go through that edge are removed from the search space for the exhaustive search phase. Each time when an edge is removed, large number of tours are removed from the search space. Although the complexity of finding a true locally optimal tour is still open, and we even do not know any nontrivial upper bounds on the number of iterations that may be needed to reach local optimality [27, 28], decades of empirical evidence and practical research have found that heuristic local search converges quickly, within low order polynomial time [1, 8, 27, 29]. In practice, we are rarely able to find perfect locally optimal tour because we simply do not allow the local search process to run enough long time. Usually we let a local search process run a predefined number of iterations, accept whatever tour it generates, and treat it as a locally optimal tour. Therefore, the size of the constructed solution attractor depends not only on the problem structure and the neighborhood function, but also on the amount of search time invested in the local search process. As we increase local search time, we will constructe a smaller and stronger solution attractor. The local search phase in the ABSS can significantly

reduce the search space for the exhaustive search phase by excluding a large number of edges. Usually the local search phase can remove about 60% of edges of the matrix E in $O(n^2)$.

Now an essential question is naturally raised: What is the relationship between the size of the constructed solution attractor and the size of the problem instance? Unfortunately, there is no theoretical analysis tool available in the literature that can be used to answer this question. We have to depend on empirical results to lend some insights. We conducted several experiments to observe the relationship between the size of the constructed solution attractor and the TSP instance size. **Figures 8–10** show the results of one of our experiments. All other similar experiments reveal the same pattern. In this experiment, we generated 10 unimodal TSP instances in the size from 1000 to 10000 nodes with 1000-node increment. For

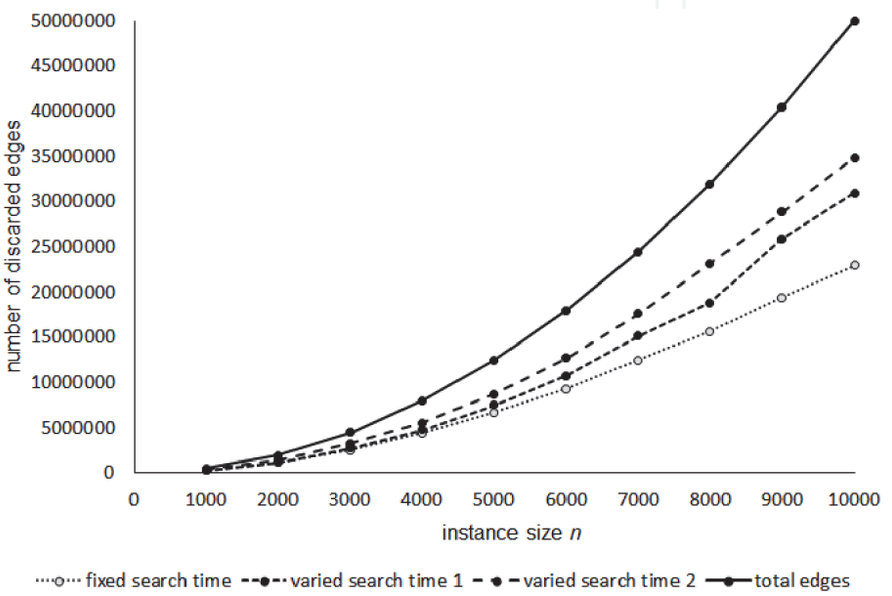


Figure 8.
The number of discarded edges at the end of local search phase.

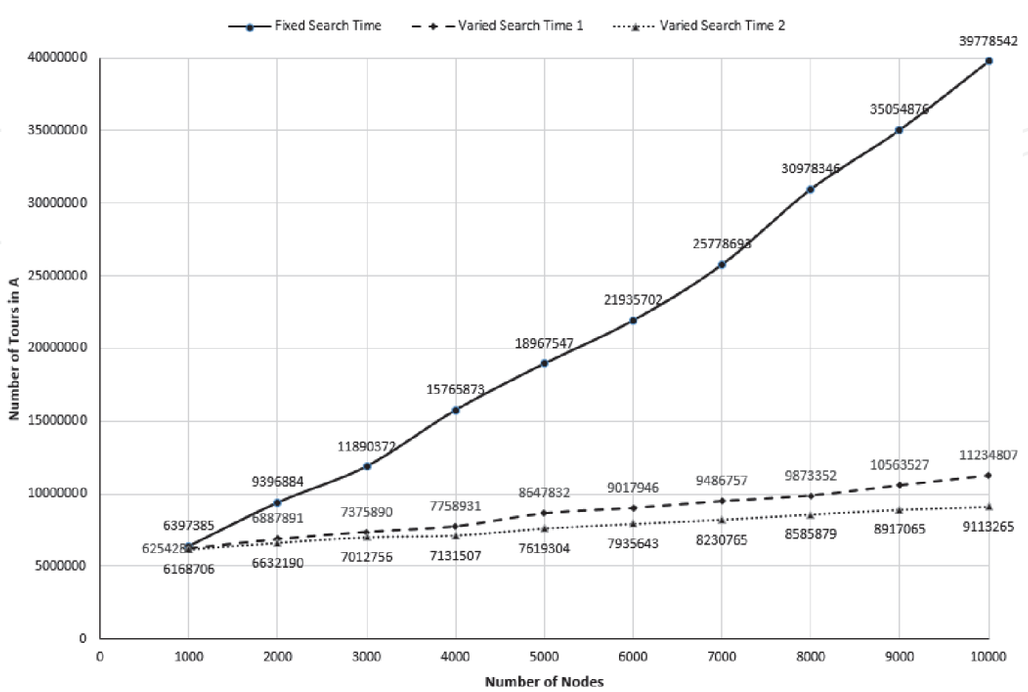


Figure 9.
Relationship between the size of the constructed solution attractor and instance size.

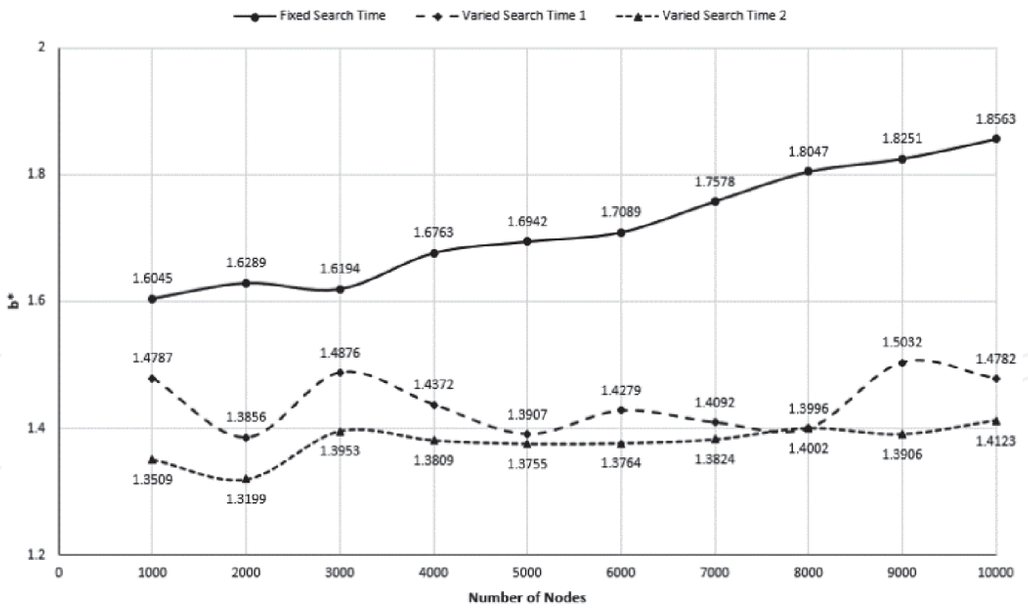


Figure 10.
The b^* values for different instance size n *in our experiment.*

each instance, the ABSS generated $K = 6n$ search trajectories. We first let each search trajectory stop when no tour improvement was made during 10000 iterations regardless of the size of the instance (named "fixed search time"). Then we did the same search procedures on these instances again. This time we made each search trajectory stop when no improvement was made during $10n$ iterations (named "varied search time 1") and $100n$ iterations (named "varied search time 2") respectively. **Figure 8** shows the number of the edges that were discarded at the end of local search phase. **Figure 9** shows the number of tours in the constructed solution attractor for each instance, and **Figure 10** shows the effective branching factors in the exhaustive search phase.

In **Figure 8**, we can see that the search trajectories can quickly converge to a small set of edges. In the fixed-search-time case, about 60% of the edges were discarded by search trajectories for the 1000-node instance, but this percentage decreases as instance size increases. For the 10000-node instance, only about 46% of the edges are discarded. However, if we increase the local search time linearly when the instance size increases, we can keep the same percentage of discarded-edge for all instance sizes. In the varied-search-time-1 case, about 60% of the edges are abandoned for all different instance sizes. In the varied-search-time-2 case, this percentage increases to 68% for all instances. Higher percentage of abandoned edges means that majority of the branches are removed from the search tree.

Figure 9 shows the number of tours exist in the constructed solution attractor for these instances. All curves in the chart appear to be linear relationship between the size of constructed solution attractor and the size of the problem instance, and the varied-search-time curves have much flatter slope because longer local search time makes a smaller constructed solution attractor. **Figures 8** and **9** indicate that the search trajectories in the local search phase can effectively and efficiently reduce the search space for the exhaustive search, and the size of the solution attractor increases linearly as the size of the problem instance increases. Therefore, the local search phase in the ABSS is an efficiently asymptotical search process that produces an extremely small search space for further exhaustive search.

The completely searching of the constructed solution attractor is delegated to the exhaustive search phase. This phase may still need to examine tens or hundreds of millions of tours but nothing a computer processor cannot handle, as opposed to the

huge number of total possibilities in the solution space. The exhaustive search phase can find the exact globally optimal tours for the problem instance after a limited number of search steps.

The exhaustive search phase can use any enumerative technique. However, the edge configuration of E can be easily searched by the depth-first tree search algorithm. One of the advantages of depth-first tree search is less memory requirement since only the nodes on the current path are stored. When using tree-search algorithm, we usually use branching factor, average branching factor, or effective branching factor to measure the computing complexity of the algorithm [30–33]. In the data structure of search tree, the branching factor is the number of successors generated by a given node. If this value is not uniform, an average branching factor can be calculated. An effective branching factor b^* is the number of sucessors generated by a typical node for a given tree-search problem. We use the following definition to calculate effective brancing factor b^* for the exhaustive search phase:

$$N = b^* + (b^*)^2 + \ldots + (b^*)^n \tag{11}$$

where n is the size of the TSP instance, representing the depth of the tree, and N is total number of nodes generated in the tree from the origin node. In our experiments, the tree-search process always starts from node 1 (the first row of E). N is total number of nodes that are processed to construct all valid tours and incomplete (therefore abandoned) tours in E. N does not count the node 1 (the origin node), but includes the node 1 as the end node of a valid tour. We use **Figure 2(d)** as an example. The depth-first search process searches the edge configuration of E and will generate $N = 58$ nodes. Therefore, $b^* \approx 1.3080$, that is, $58 \approx 1.3080 + 1.3080^2 + \ldots + 1.3080^{10}$. **Figure 10** shows the effective branching factor b^* in our experiment. The low values of b^* indicates that the edge configuration of the solution attractor represents a tree with extremely sparse branches, and the degree of sparseness does not changes as the problem size increase if we linearly increase local search time in the local search phase for a large instance. The search time in the exhaustive search phase is probably in $O(n^2)$ since the size of the constructed solution attractor might be linearly increased with the problem size n and the number of edges in E is polynomially increased with the problem size. Our experiments shows that the ABSS can significantly reduce the computational complexity for the TSP and solve the TSP efficiently with global optimality guarantee.

Therefore, the ABSS is a simple algorithm that increases in computational difficulty polynomially with the size of the TSP. In the ABSS, the objective pursued by the local search phase is "quickly eliminating unnecessary search space as much as possible." It can provide an answer to the question "In which small region of the solution space is the optimal solution located?" in time of $O(n^2)$. The objective of the exhaustive search phase is "identifying the best tour in the remaining search space." It can provide an anwer to the question "Which is the best tour in this small region?" in time of $O(n^2)$. All together, the ABSS can answer the question "Is this tour the best tour in the solution space?" in time of $O(n^2)$. Therefore, the ABSS is probably with computing complexity of $O(n^2)$ and memory space requirement of $O(n^2)$. This suggests that the TSP might not be as complex as we might have expected.

7. Conclusion

Advances in computational techniques on the determination of the global optimum for an optimization problem can have great impact on many scientific and

engineering fields. Although both the TSP and heuristic local search algorithms have huge literature, there is still a variety of open problems. Numerous experts have made huge advance on the TSP research, but two fundamental questions of the TSP remain essentially open: "How can we find the optimal tours in the solution space, and how do we know they are optimal?"

The P-vs-NP problem is about how fast we can search through a huge number of solutions in the solution space [34]. Do we ever need to explore all the possibilities of the problem to find the optimal one? Actually, the P-vs-NP problem asks whether, in general, we can find a method that completely searches only the region where the optimal points are located [34–36]. Most people believe P $\neq$ NP because we have made little fundamental progress in the area of exhaustive search. Modern computers can greatly speed up the search, but the extremely large solution space would still require geologic search time to find the exact optimal solution on the fastest machines imaginable. A new point of view is needed to improve our capacity to tackle these difficulty problems. This paper describe a new idea: using efficient local search process to effectively reduce the search space for exhaustive search. The concept of solution attractor in heuristic local search systems may change the way we think about both local search and exhaustive search. Heuristic local search is an efficient search system, while exhaustive search is an effective search system. The key is how we combines these two systems into one system beautifully to conquer the fundamental issues of the hard optimization problems. In the TSP case, the edge matrix E, a problem-specific data structure, plays a critical role of reducing the search space and transforming local search to global search.

The ABSS is designed for the TSP. However, the concepts and formulation behind the search algorithm can be used for any combinatorial optimization problem requiring the search of a node permutation in a graph.

Author details

Weiqi Li
School of Management, University of Michigan-Flint, Flint, USA

*Address all correspondence to: weli@umich.edu

References

[1] Applegate DL, Bixby RE, Chaátal V, Cook WJ. The Traveling Salesman Problem: A Computational Study. Princeton: Princeton University Press; 2006

[2] Papadimitriou CH, Steiglitz K. Combinatorial Optimization: Algorithms and Complexity. New York: Dover Publications; 1998

[3] Papadimitriou CH, Steiglitz K. On the complexity of local search for the traveling salesman problem. SIAM Journal on Computing. 1977:6:76–83

[4] Gomey J. Stochastic global optimization algorithms: a systematic formal approach. Information Science. 2019:472:53–76

[5] Korte B, Vygen J. Combinatorial Optimization: Theory and Algorithms. New York: Springer; 2007

[6] Rego C, Gamboa D, Glover F, Osterman C. Traveling salesman problem heuristics: leading methods, implementations and latest advances. European Journal of Operational Research. 2011:211:427–411

[7] Zhigliavsky A, Zillinakas A. Stochastic Global Optimization. New York: Springer; 2008

[8] Aart E, Lenstra JK. Local Search in Combinatorial Optimization. Princeton: Princeton University Press; 2003

[9] Michalewicz Z, Fogel DB. How to Solve It: Modern Heuristics. Berlin: Springer; 2002

[10] Sahni S, Gonzales T. P-complete approximation problem. Journal of the ACM. 1976:23:555–565

[11] Sourlas N. Statistical mechanics and the traveling salesman problem. Europhysics Letters. 1986:2:919–923

[12] Savage SL. Some theoretical implications of local optimality. Mathematical Programming. 1976:10: 354–366

[13] Shammon CE. A mathematical theory of communication. Bell System Technical Journal. 1948:27:379–423& 623–656

[14] Alligood KT, Sauer TD, York JA. Chaos: Introduction to Dynamical System. New York: Springer; 1997

[15] Auslander J, Bhatia NP, Seibert P. Attractors in dynamical systems. NASA Technical Report NASA-CR-59858; 1964

[16] Brin M, Stuck G. Introduction to Dynamical Systems. Cambridge: Cambridge University Press

[17] Brown R. A Modern Introduction to Dynamical Systems. New York: Oxford University Press.

[18] Denes A, Makey G. Attractors and basis of dynamical systems. Electronic Journal of Qualitative Theory of Differential Equations. 2011:20(20):1–11

[19] Fogedby H. On the phase space approach to complexity. Journal of Statistical Physics. 1992:69:411–425

[20] Milnor J. On the concept of attractor. Communications in Mathematical Physics. 1985:99(2):177–195

[21] Milnor J. Collected Papers of John Milnor VI: Dynamical Systems (1953–2000). American Mathematical Society; 2010

[22] Ruelle D. Small random perturbations of dynamical systems and the definition of attractor. Communications in Mathematical Physics. 1981:82:137–151

[23] Li W. Dynamics of local search trajectory in traveling salesman problem. Journal of Heuristics. 2005:11: 507–524

[24] Li W, Feng M. Solution attractor of local search in traveling salesman problem: concepts, construction and application. International Journal of Metaheuristics. 2013:2(3): 201–233

[25] Li W, Li X. Solution attractor of local search in traveling salesman problem: computational study. International Journal of Metaheuristics. 2019:7(2):93–126

[26] Kroese DP, Taimre T, Botev ZI. Handbook of Monte Carlo Methods. New York: John Wiley & Sons; 2011

[27] Fischer ST. A note on the complexity of local search problems. Information Processing Letters. 1995:53 (2):69–75

[28] Chandra B, Karloff HJ, Tovey CA. New results on the old-opt algorithm for the traveling salesman problem. SIAM Journal on Computing. 1999:28(6): 1998–2029

[29] Grover, LK. Local search and the local structure of NP-complete problems. Operations Research Letters. 1992:12(4):235–243

[30] Baudet GM. On the branching factor of the alpha-beta pruning algorithm. Artificial Intelligence. 1978: 10(23):173–199

[31] Edelkamp S, Korf RE. The branching factor of regular search space. Proceedings of the 15th National Conference on Artificial Intelligence. 1998:292–304

[32] Korf RE. Depth-first iterative deepening: an optimal admissible tree search. Artificial Intelligence. 1985:27: 97–109

[33] Pearl J. The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. Communication of the ACM. 1982:25 (8):559–564

[34] Fortnow L. The Golden Ticket – P, NP, and the Search for the Impossible. Princeton: Princeton University Press; 2013

[35] Fortnow L. The status of the P versus NP problem. Communication of the ACM. 2009:52(9):78–86

[36] Sipser M. The history of status of the P verses NP question. Proceedings of 24th Annual ACM Symposium on Theory of Computing. 1992:603–618